

Understanding and Improving Service Discovery in Hostile and Volatile Environments



Chris Dabrowski (897)

Jesse Elder (897)

Kevin Mills (892)

Doug Montgomery (892)

Scott Rose (892)

**NRC Review Panel
February 8, 2002**



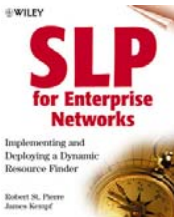

“Agile Software for an Uncertain World”

Dynamic Discovery Protocols in Essence

Dynamic discovery protocols enable *network elements* (including software clients and services, as well as devices):

- (1) to *discover* each other without prior arrangement,
- (2) to *express* opportunities for collaboration,
- (3) to *compose* themselves into larger collections that cooperate to meet an application need, and
- (4) to *detect and adapt to changes* in network topology.

Selected First-Generation Dynamic Discovery Protocols

| | | |
|---|---|---|
|  <p>3-Party Design</p> |  <p>2-Party Design</p> |  <p>Adaptive 2/3-Party Design</p> |
|  <p>Vertically Integrated 3-Party Design</p> |  <p>Network-Dependent 3-Party Design</p> |  <p>Network-Dependent 2-Party Design</p> |

Presentation Outline

- Project objective and plan (one slide)
- Accomplishments over past year (one slide)
- Context setting (two slides)
 - Alternative architectures for service-discovery systems
 - Technical approach to modeling and analysis in first phase of the project
- Understanding consistency-maintenance in service-discovery systems under increasing communication failure (12 slides)
- Summary of progress to date (one slide)
- Plans for upcoming year (one slide)
- Addenda
 - Additional details on consistency-maintenance experiments
 - Issues relayed to Sun after correctness analysis of Jini specification
 - Performance measurement results from Jini and UPnP implementations

Project Objective

Research, design, and characterize self-adaptive mechanisms to improve performance of discovery protocols in hostile and volatile environments.

Project Plan in Three Phases

Phase I – characterize performance of selected service-discovery protocols (Universal Plug-and-Play – UPnP – and Jini) as specified and implemented

- develop architectural models for each protocol
- establish performance benchmarks based on default or recommended parameter values and on required or most likely implementation of behaviors

Phase II – design, simulate, and characterize self-adaptive mechanisms to improve performance in volatile environments

- devise algorithms to adjust control parameters and behavior in each protocol
- simulate performance of each algorithm against benchmark performance
- select most promising algorithms for further development

Phase III – implement and validate the most promising algorithms in publicly available reference software (from Sun Microsystems and Intel)

Accomplishments This Year

Modeling

- Completed Rapide architectural-description language model of UPnP specification (Jini specification completed last year)
- Developed framework of (SLX) discrete-event simulation model for UPnP (model patterned after publicly available Intel Linux implementation)

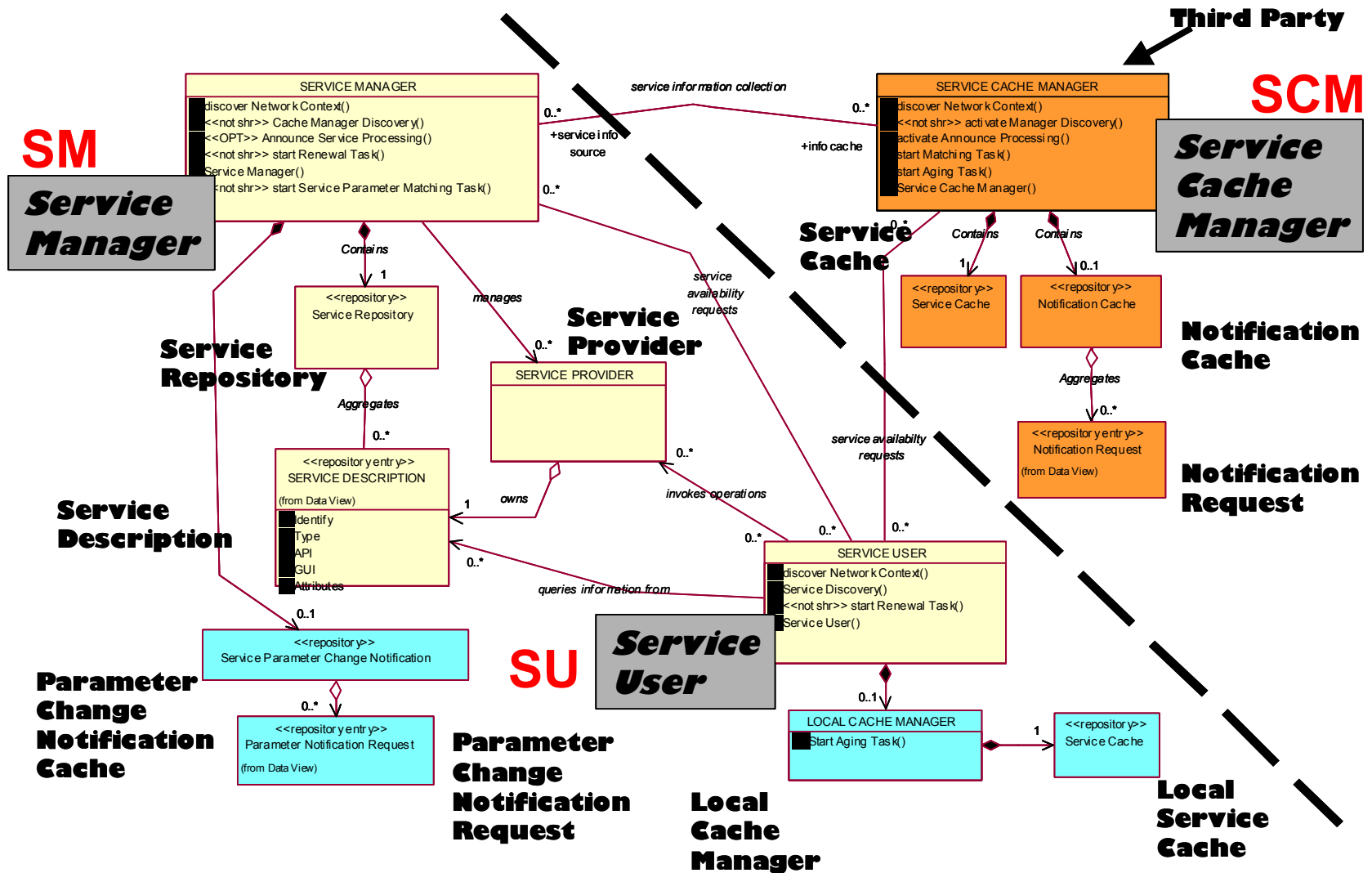
Analysis

- Explored correctness of Jini model against various consistency conditions; found four areas of concern; briefed Sun's Jini design team on findings (see addendum for more information)
- Investigated consistency maintenance in Jini and UPnP models under increasing interface-failure rate (technical subject of current talk)

Measurement (see addendum for more information)

- Designed independent benchmark service and scenarios for service-discovery systems
- Created synthetic workload generation tools for emulating the behavior of large scale, dynamic, ad-hoc networking environments
- Conducted baseline performance measurement experiments to compare the performance of UPnP and Jini in benchmark scenarios

Two Party vs. Three Party Architectures



Phase I Modeling Approach: Use Rapide to Model and Understand Dynamics of Jini and UPnP

| Time | Command | Parameters |
|------|-------------|------------------------------|
| 5 | NodeFail | SM4 |
| 5 | LinkFail | SCM1 SM4 |
| 10 | GroupJoin | SM4 GROUP1 |
| 10 | FindService | SU8 5 1 2 S XYZ ALL |
| 50 | AddService | SM4 SCM3 T ATT API GUI 20 30 |

Scenario

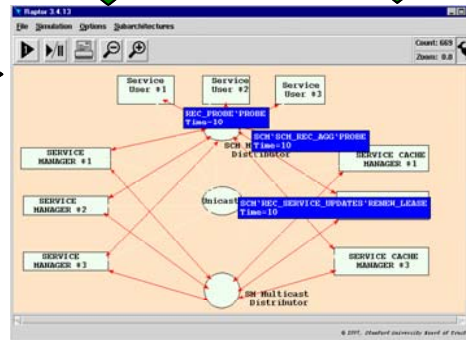
Topology

Specification Model

```

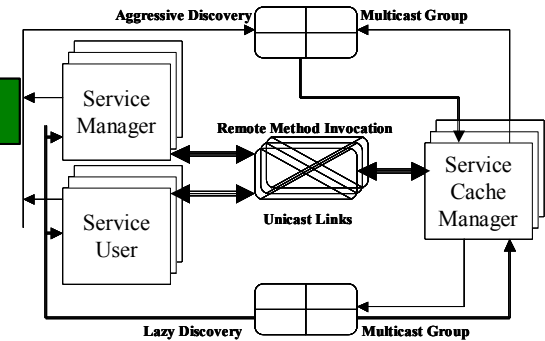
-- *****
-- ** 3.3 DIRECTED DISCOVERY CLIENT INTERFACE **
-- *****
-- This is used by all JINI entities in directed
-- discovery mode. It is part of the SCM_Discovery
-- Module. Sends Unicast messages to SCMs on list of
-- SCMS to be discovered until all SCMS are found.
-- Receives updates from SCM DB of discovered SCMs and
-- removes SCMs accordingly
-- NOTE: Failure and recovery behavior are not
-- yet defined and need review.
TYPE Directed_Discovery_Client
(SourceID : IP_Address; InSCMsToDiscover : SCMList; StartOption : DD_Code;
 InRequestInterval : TimeUnit; InMaxNumTries : integer; InPV : ProtocolVersion)
IS INTERFACE
SERVICE DDC_SEND_DIR : DIRECTED_2_STEP_PROTOCOL;
SERVICE DISC_MODES : dual SCM_DISCOVERY_MODES;
SERVICE DD_SCM_Update : DD_SCM_Update;
SERVICE SCM_Update : SCM_Update;
SERVICE DB_Update : dual DB_Update;
SERVICE NODE_FAILURES : NODE_FAILURES; -- events for failure and recovery.
ACTION
IN Send_Requests(),
BeginDirectedDiscovery();
BEHAVIOR
action animation_Iam (name: string);
MySourceID : VAR IP_Address;
PV : VAR ProtocolVersion;

```

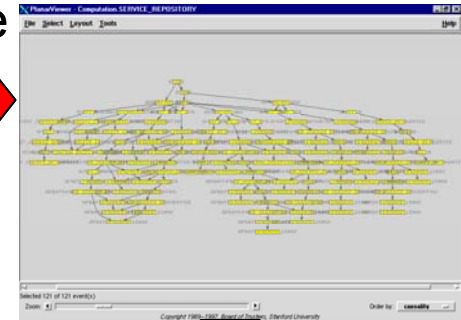


Consistency Conditions

- For All (SM, SD, SCM):
(SM, SD) IsElementOf SCM registered-services
implies SCM IsElementOf SM discovered-SCMs (CC1)
- For All (SM, SD, SCM):
SCM IsElementOf SM discovered-SCMs &
(SD) IsElementOf SM managed-services
implies (SM, SD) IsElementOf SCM registered-services (CC2)
- For All (SM, SD, SCM):
SCM IsElementOf SM discovered-SCMs &
(SM, SD) IsElementOf SCM registered-services &
NOT (SCM IsElementOf SM persistent-list)
implies Intersection (SM GroupsToJoin, SCM GroupsMemberOf) (CC3)
- For All (SM, SD, SCM, SU, NR):
(SU, NR) IsElementOf SCM requested-notifications &
(SM, SD) IsElementOf SCM registered-services &
Matches((SM, SD), (SU, NR))
implies (SM, SD) IsElementOf SU matched-services (CC4)



Execute with Rapide



Analyze POSETs

Assess Correctness, Performance, & Complexity

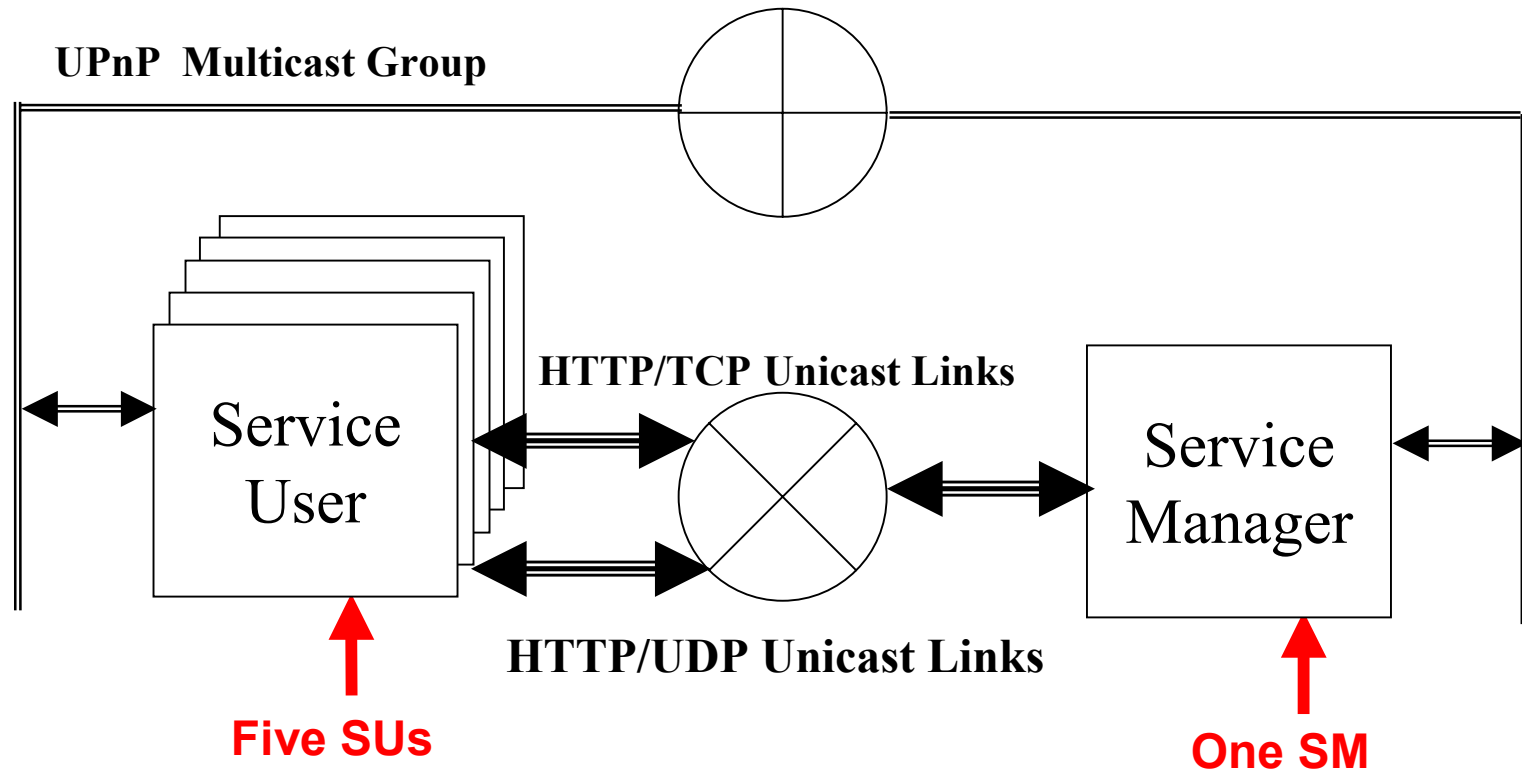
Understanding Consistency Maintenance in Discovery Architectures during Interface Failure

How do various service discovery architectures, topologies, and fault-recovery mechanisms perform under deadline during interface failure?

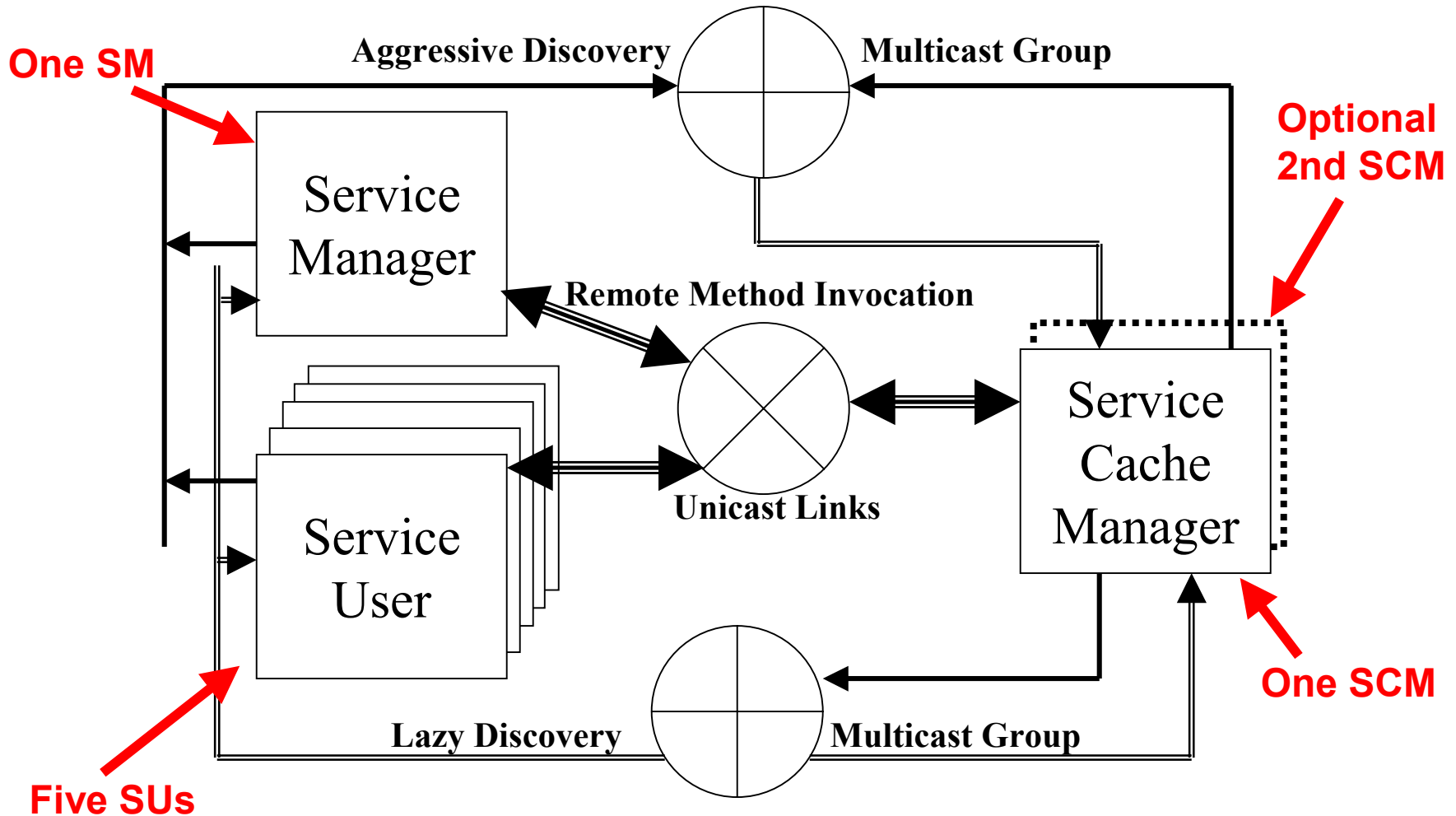
Outline of Experiment

- Deploy models of two-party (UPnP) and three-party (Jini) architectures with one SM and five SUs (for Jini include two topologies – one and two SCMs).
- Ensure initial discovery and information propagation completed.
- Introduce a change in the service description at the SM, and establish a deadline for propagating the new information to all SUs.
- Measure the number of messages exchanged and the latency required to propagate the information to all SUs, or until the deadline arrives, under two different propagation mechanisms: polling and eventing.
- Repeat this experiment while varying the percentage of interface failure time for each node up to 75% (in increments of 5%).

Two-Party (UPnP) Topology for Experiment



Three-Party (Jini) Topologies for Experiment



Failures Interfere with Discovery and Information Propagation

- Interface Failure – Tx, Rx, or Both
 - Due to nearby enemy jamming or other interference
 - Due to multi-path fading during mobility
- Path Loss – Pt-Pt or Area-Area and Full-duplex or Half-duplex
 - Due to persistent congestion
 - Due to physical link cuts
 - Due to enemy jamming at routers
- Message Loss – under both UDP and TCP (>delay)
 - Due to sporadic or distant enemy jamming or other interference
 - Due to transient congestion
 - Due to multi-path fading during mobility
- Node Failure – Partial or Complete with variable persistence of information
 - Due to enemy bombardment or cyber attacks
 - Due to mobility associated with military operations

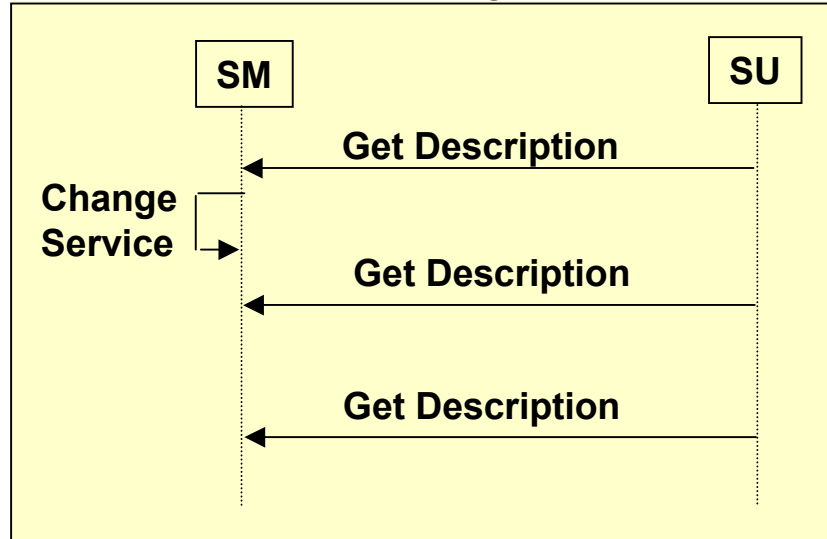
Discovery Systems Divide Recovery Responsibilities: Lower Layers, Discovery Protocol, and Application

- Selective Reliable Delivery by Lower Layers
 - TCP attempts retransmissions (basis for Jini-RMI and UPnP-HTTP/TCP)
 - UDP messages in UPnP sent as multiple copies
- Periodic Announcements by Discovery Protocol
 - Allows caching nodes to discard information when TTL expires
 - Jini includes aggressive search at node start up, while UPnP permits nodes to undertake aggressive search at any time
- Periodic Refreshing of Resources Required by Discovery Protocol
 - Allows resource owner to free resource when refresh period expires
- Remote Exceptions Issued by Protocol Over TCP Links
 - Allows application to take recovery action: Ignore? Retry? Discard knowledge of service or resource?

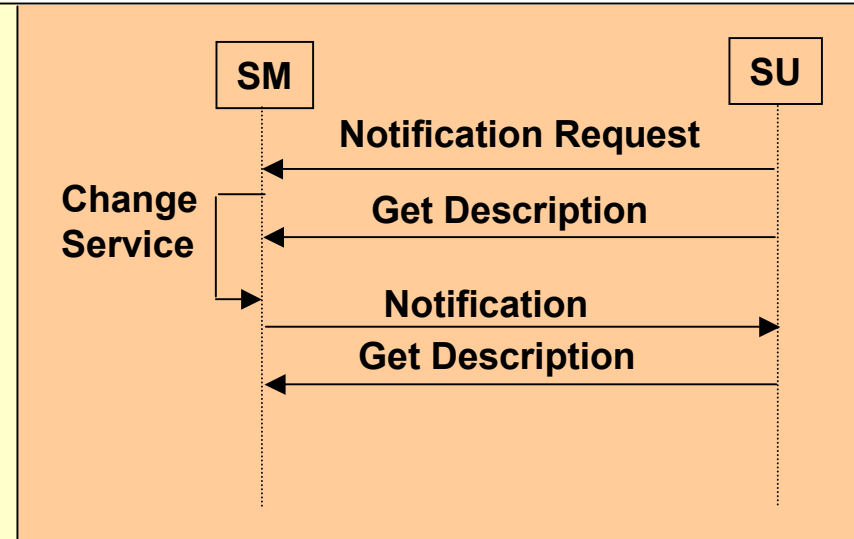
Consistency-Maintenance Mechanisms for Experiment

Two Party

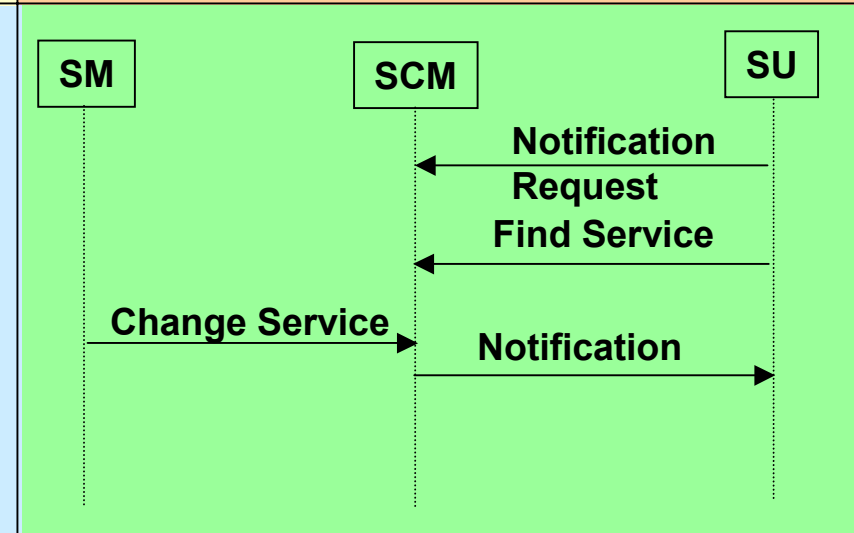
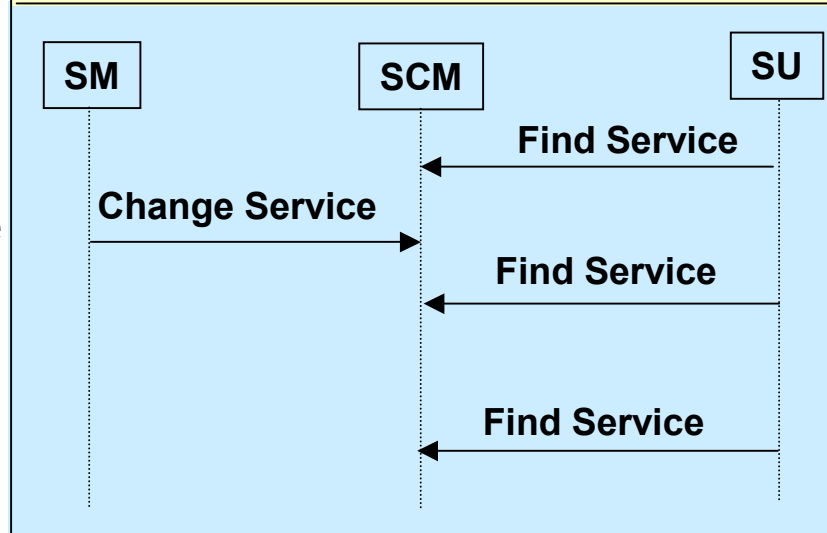
Polling



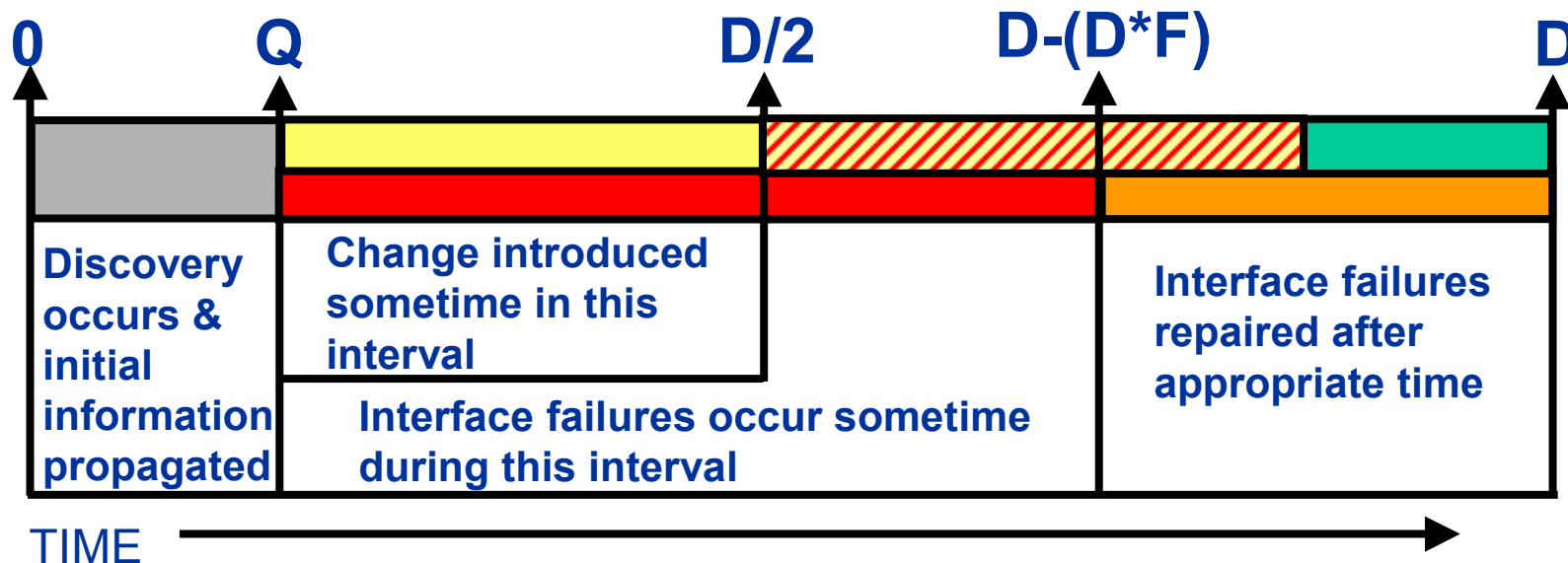
Eventing



Three Party



Interface-Failure Model for Experiment



- Random Processes
1. Choose a time to introduce the change [uniform(Q , $D/2$)]
 2. For each node, choose a time to introduce an interface failure [uniform(Q , $D-(D*F)$)]
 3. When each interface failure occurs, choose the scope of the failure, where each of [Rx, Tx, Both] has an equal probability

Q = end of quiescent period (100 s in our experiment)

D = propagation deadline (5400 s in our experiment)

F = Interface Failure Rate (variable from 0% - 75% in 5% increments in our experiment)

Our Modeled Responses to Remote Exceptions

(“Approximate” – see addendum for details)

- Ignore REX Received
 - When replying to a remote-method invocation
 - When attempting to cancel a lease
 - When attempting to renew a lease (But then attempt to obtain a new lease)
- Retry Operation for Some Period of Time - Then Quit If Not Successful
 - When attempting to register for notification events
 - When a UPnP SU requests service descriptions
 - When Jini SM attempts to register service or change service on SCM
- Eliminate Local Knowledge of Discovered Entity
 - When remote exceptions occur for too long a period
 - When UPnP SU fails to hear scheduled SM announcement

Metrics Devised for Consistency-Maintenance Experiments

(see addendum for formal definitions)

- Update Responsiveness (R)

Assuming that information is created at a particular time and must be propagated by a deadline, then the difference between the deadline and the creation time represents available time in which to propagate the information. Update Responsiveness, R , measures the *proportion of the available time remaining after the information is propagated*. [1 = all time remains and 0 = no time remains]

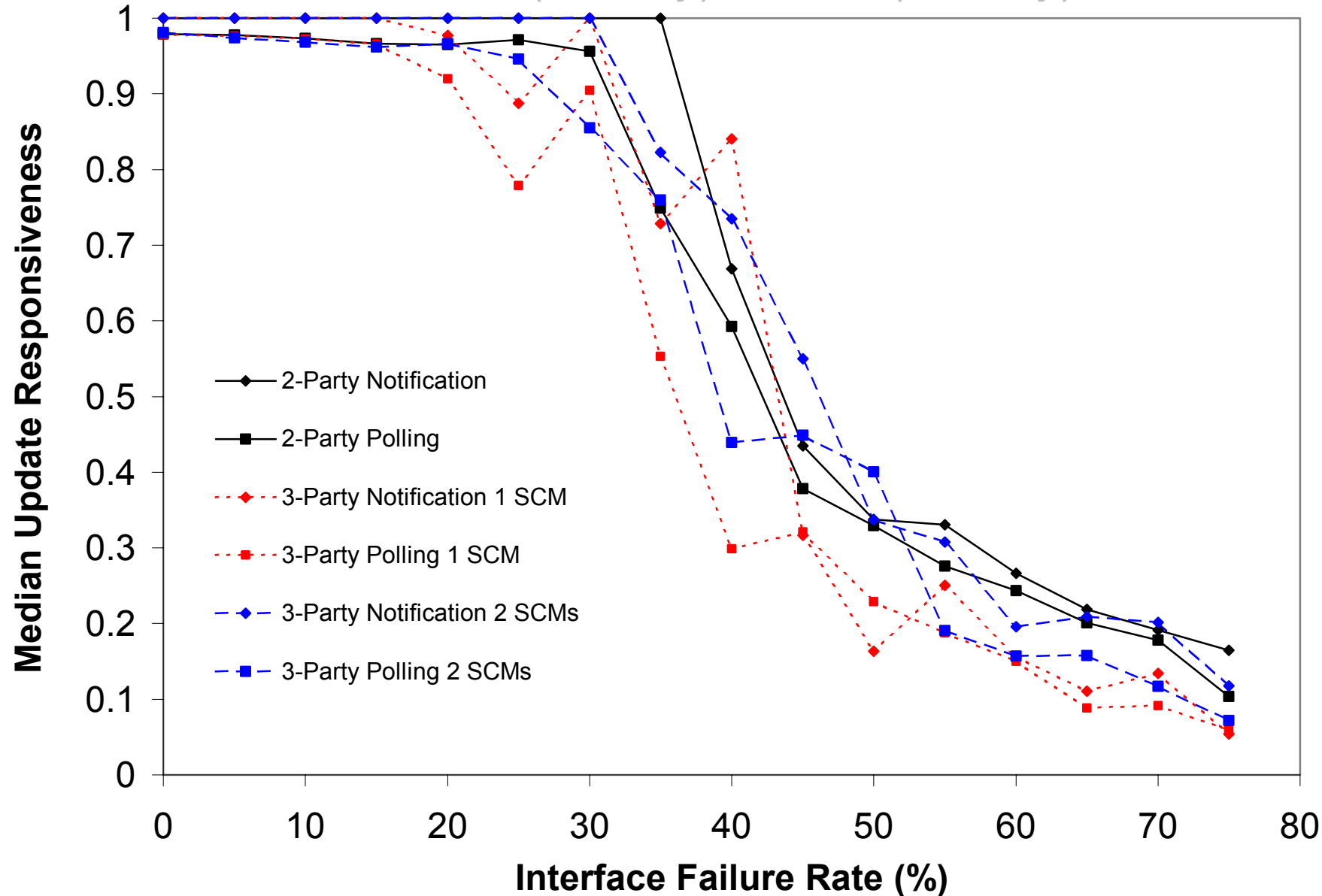
- Update Effectiveness (U)

Update Effectiveness, U , measures the *probability that information will propagate successfully to a SU before some deadline, D* . [1 = information will be propagated and 0 = information will not be propagated]

- Update Efficiency (E)

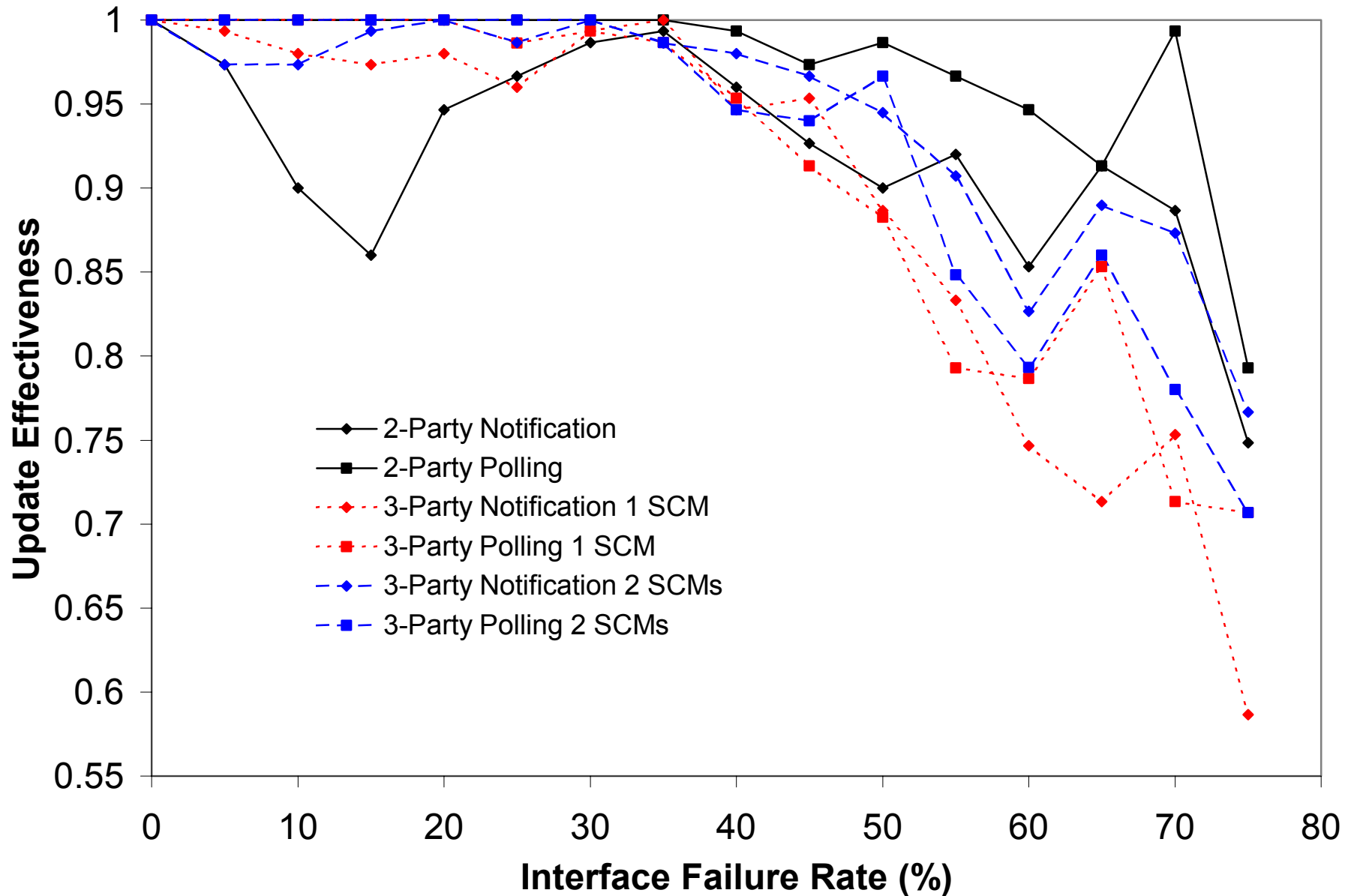
Given a specific topology of SUs and SMs in a discovery system, examination of the available architectures (two-party and three-party) and mechanisms (polling and eventing) will reveal a minimum number of messages that need to be sent to propagate information from all SMs to all SUs in the topology. Update Efficiency, E , can be measured as the *ratio of the minimum number of messages needed to the actual number of messages observed*. [1 = only minimum number of messages needed and 0 = infinite number of messages needed]

Median Update Responsiveness UPnP (2-Party) vs. Jini (3-Party)

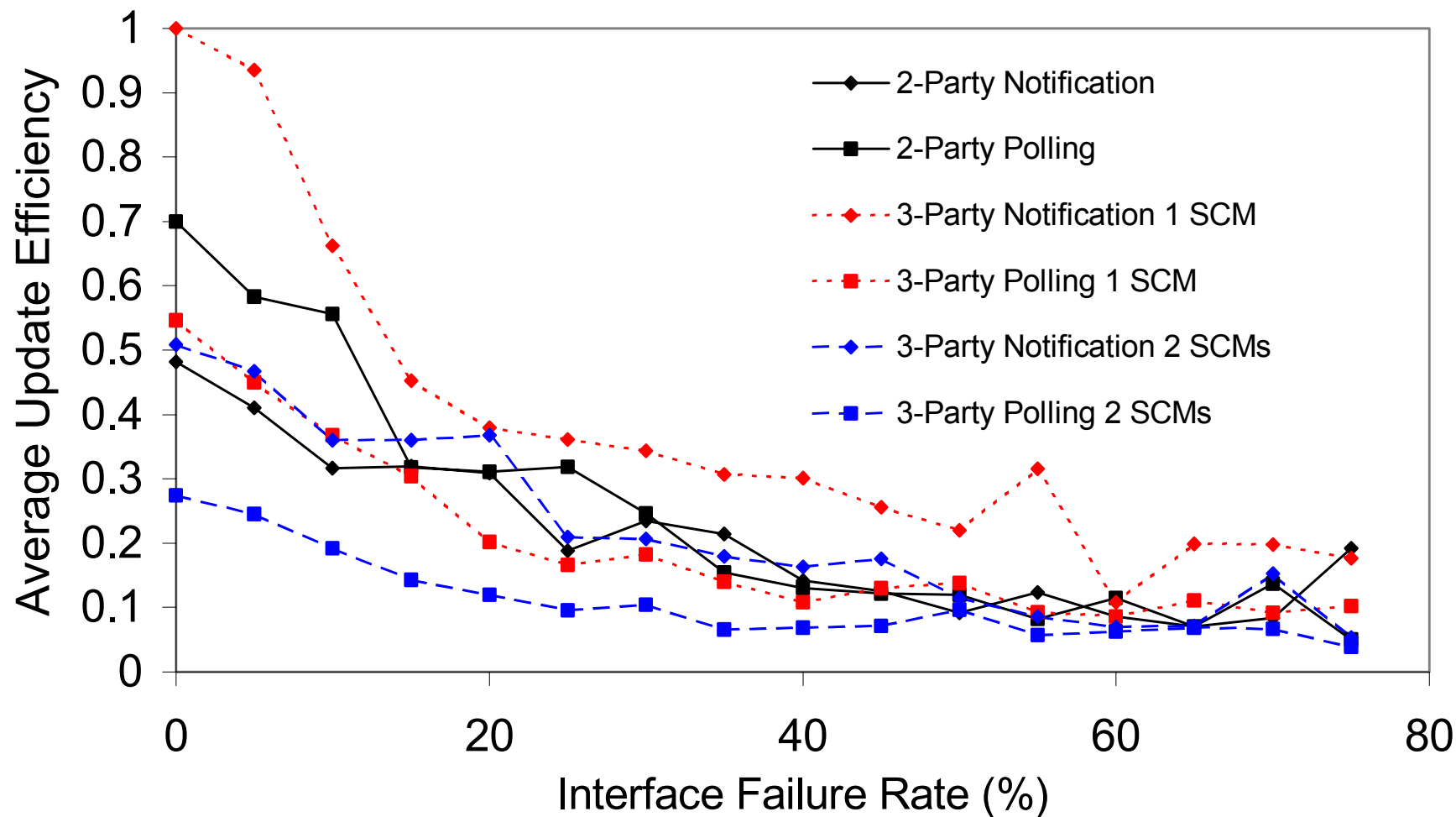


Update Effectiveness

UPnP (2-Party) vs. Jini (3-Party)



Average Update Efficiency UPnP (2-Party) vs. Jini (3-Party)



Summary of Progress to Date

- Models of service-discovery protocols (written in the Rapide architecture-description language) :
 - Jini (models the Sun Microsystems version 1.1 specification)
 - UPnP (models the Microsoft version 1.0 specification)
- Papers
 - “Analyzing Properties and Behavior of Service Discovery Protocols Using an Architecture-based Approach”, C. Dabrowski and K. Mills, *Proceedings of Working Conference on Complex and Dynamic Systems Architectures*
 - “Understanding Consistency Maintenance in Service Discovery Architectures during Communication Failure”, C. Dabrowski, K. Mills, and J. Elder, submitted to the *3rd International Workshop on Software Performance*.
 - “Survivable Software for Harsh Environments”, C. Dabrowski, K. Mills, and J. Elder, submitted to *MILCOM 2002*.
- Interactions
 - Gave an invited “Tech-a-Tech” presentation to Sun Jini team
 - Presented work at 2001 Pervasive Computing Conference
 - Obtained co-funding from DARPA and ARDA

Plans for Upcoming Year

Modeling

- Complete (SLX) discrete-event simulation model for UPnP
- Validate model against current results from Rapide model
- Develop SLX simulation models for Jini and SLP
- Extend generic structural model of service discovery protocols to include message exchanges and consistency conditions, and publish the extended generic model

Analysis

- Investigate consistency maintenance in Jini and UPnP models under increasing message-loss rate
- Investigate consistency maintenance in Jini and UPnP models under node and thread failure
- Propose, model, and evaluate selected self-adaptive mechanisms for UPnP
- Consider analytical model for update effectiveness during interface failure

***Slides Containing Additional Details
on Consistency-Maintenance Experiments***

Equating a Generic Structural Model of Service Discovery Architectures to Selected Commercial Discovery Systems

| Generic Model | Jini | UPnP | SLP |
|----------------------|-------------------------|----------------------------|------------------------------------|
| Service User | Client | Control Point | User Agent |
| Service Manager | Service or Device Proxy | Root Device | Service Agent |
| Service Provider | Service | Device or Service | Service |
| Service Description | Service Item | Device/Service Description | Service Registration |
| Identity | Service ID | Universal Unique ID | Service URL |
| Type | Service Type | Device/Service Type | Service Type |
| Attributes | Attribute Set | Device/Service Schema | Service Attributes |
| User Interface | Service Applet | Presentation URL | Template URL |
| Program Interface | Service Proxy | Control/Event URL | Template URL |
| Service Cache Manger | Lookup Service | not applicable | Directory Service Agent (optional) |

The Six Combinations of Architecture, Topology, and Consistency-Maintenance Mechanism Used in Experiments

| Architectural Variant | Protocol Basis | Consistency-Maintenance Mechanism |
|--------------------------|----------------|---|
| Two-Party | UPnP | Polling |
| Two-Party | UPnP | Notification (with notification registration on SM) |
| Three-Party (Single SCM) | Jini | Polling (with service registration on SCM) |
| Three-Party (Single SCM) | Jini | Notification (with service registration and notification registration on SCM) |
| Three-Party (Dual SCM) | Jini | Polling (with service registration on SCM) |
| Three-Party (Dual SCM) | Jini | Notification (with service registration and notification registration on SCM) |

Specific Division of Failure-Recovery Responsibilities Used in Experiments

| Responsible Party | Recovery Mechanism | Two-Party Architecture (UPnP) | Three-Party Architecture (Jini) |
|-----------------------|----------------------|---|---|
| Lower-Layer Protocols | UDP | No recovery | No recovery |
| | TCP | Issue REX in 30-75 s | Issue REX in 30-75 s |
| Discovery Protocols | Lazy Discovery | SM: announces with $n(3+2d+k)$ messages every 1800 s | SCM: announces every 120 s |
| | Aggressive Discovery | SU: issues <i>Msearch</i> every 120 s (after purging SD) | SU and SM: issue seven probes (at 5 s intervals) only during startup |
| Application Software | Ignore REX | SU: <i>HTTP Get</i> Poll SM: Notification | SU: <i>FindService</i> Poll SCM: Notification |
| | Retry after REX | SU: <i>HTTP Get</i> after discovery retry in 180 s (retries ≤ 3) <i>Subscribe</i> requests retry in 120s | SM: depositing or refreshing SD copy on SCM retry in 120s SU: registering and refreshing notification requests with SCM retry in 120 s |
| | Discard Knowledge | SU: purge SD after failure to receive SM announcement within 1800 s | SU and SM: purge SCM after 540 s of continuous REX |

Significant Parameters and Values Used in Experiments

| | Parameter | Value |
|---|---|--|
| Behavior in both two- and three-party architectures | Polling interval | 180 s |
| | Registration TTL | 1800 s |
| | Time to retry after REX (if applicable) | 120 s |
| UPnP-specific behavior for two-party architecture | Announce interval | 1800 s |
| | Msearch query interval | 120 s |
| | SU purges SD | At TTL expiration |
| Jini-specific behavior for three-party architecture | Probe interval | 5 s (7 times) |
| | Announce interval | 120 s |
| | SM or SU purges SCM | After 540 s with only REX |
| Interface failure parameters | Failure incidence | Once per run for each node |
| | Failure scope | Transmitter, receiver, or both with equal likelihood |
| | Failure duration | 5% increments of 5400 s from 0 to 75% |
| Transmission and processing delays | UDP transmission delay | 10 us constant |
| | TCP transmission delay | 10-100 us uniform |
| | Per-item processing delay | 100 us for cache items 10 us for other items |

Console Output from a Sample Experiment Run

Rate - 5

Run number - 21

SM 1 OUT Interface down 365, up 635

SCM 1 OUT Interface down 2417, up 2687

SCM 2 IN & OUT Interface down 519, up 789

SU 1 IN Interface down 2238, up 2508

SU 2 IN Interface down 3256, up 3526

SU 3 IN Interface down 207, up 477

SU 4 OUT Interface down 2876, up 3146

SU 5 IN Interface down 4478, up 4748

Performance:

SM 1 346.00000 346.00000 6 17

SCM 1 346.00000 346.00016 61 102

SCM 2 346.00000 346.00015 61 105

SU 1 346.00000 346.00109 0 11

SU 2 346.00000 346.00109 0 11

SU 3 346.00000 5400.00000 4 11

SU 4 346.00000 346.00109 0 11

SU 5 346.00000 346.00114 0 11

Update Responsiveness (R)

Let D be a deadline by which we wish to propagate information to each service user (SU) node (n) in a service discovery topology.

Let t_c be the creation time of the information that we wish to propagate, where $t_c < D$.

Let $t_{U(n)}$ be the time that the information is propagated to SU n , where $n = 1$ to N , and N is the total number of SUs in a service discovery topology.

Define information-propagation latency (L) for an SU n as:

$$L_n = (t_{U(n)} - t_c) / (\max(D, t_{U(n)}) - t_c).$$

Define update responsiveness (R) for an SU n as:

$$R_n = 1 - L_n.$$

Update Effectiveness (U)

Let the definitions related to Update Responsiveness, R , hold.

Let X represent the number of runs during which a particular service discovery topology is observed under identical conditions.

Recalling that N is the total number of SUs in a service discovery topology, define the number of SUs observed under identical conditions as:

$$O = X \cdot N.$$

Define the probability of failure to propagate information to an SU as:

$$P(F) = (\text{count}(R_{i,j} == 0))/O, \text{ where } i = 1..N \text{ and } j = 1..X.$$

Define the Update Effectiveness for a given set of conditions as:

$$U = 1 - P(F).$$

Update Efficiency (E)

Let the preceding definitions associated with Update Responsiveness and Update Effectiveness hold.

Let M be the minimum number of messages needed to propagate information from all SMs to all SUs.

Let S be the observed number of messages sent while attempting (failures may occur) to propagate information from all SMs to all SUs in a given run of the topology.

Define average Update Efficiency as:

$$E_{avg} = (\text{sum}(M/S_k))/X, \text{ where } k = 1..X.$$

Summary Statistics for Performance of Each Combination on Each Metric

| | Mean (across all interface-failure rates) | | |
|--|---|---------------|--------------------|
| | Median Responsiveness | Effectiveness | Average Efficiency |
| Two-Party Notification | 0.663 | 0.921 | 0.212 |
| Two-Party Polling | 0.615 | 0.973 | 0.251 |
| Three-Party Notification (Single SCM) | 0.601 | 0.894 | 0.389 |
| Three-Party Polling (Single SCM) | 0.530 | 0.911 | 0.201 |
| Three-Party Notification (Dual SCM) | 0.655 | 0.942 | 0.221 |
| Three-Party Polling (Dual SCM) | 0.587 | 0.927 | 0.110 |

95% C.I. for Each Metric-Combination at Selected Failure Rates

| | Responsiveness | | | Effectiveness | | | Efficiency | | |
|--|----------------|-------|-------|---------------|-------|-------|------------|-------|-------|
| | 5% | 40% | 75% | 5% | 40%. | 75% | 5% | 40% | 75% |
| Two-Party Notification | 1.000 | 0.561 | 0.111 | 0.970 | 0.954 | 0.709 | 0.354 | 0.065 | 0.031 |
| | 1.000 | 0.783 | 0.162 | 0.977 | 0.966 | 0.787 | 0.467 | 0.220 | 0.354 |
| Two-Party Polling | 0.975 | 0.501 | 0.076 | 1.000 | 0.993 | 0.760 | 0.501 | 0.031 | 0.042 |
| | 0.980 | 0.849 | 0.138 | 1.000 | 0.993 | 0.826 | 0.666 | 0.230 | 0.059 |
| Three-Party Notification (Single SCM) | 1.000 | 0.605 | 0.042 | 0.993 | 0.939 | 0.521 | 0.827 | 0.099 | 0.033 |
| | 1.000 | 1.000 | 0.095 | 0.993 | 0.955 | 0.652 | 1.000 | 0.504 | 0.320 |
| Three-Party Polling (Single SCM) | 0.974 | 0.244 | 0.043 | 1.000 | 0.946 | 0.660 | 0.387 | 0.043 | 0.040 |
| | 0.980 | 0.412 | 0.083 | 1.000 | 0.960 | 0.753 | 0.512 | 0.173 | 0.164 |
| Three-Party Notification (Dual SCM) | 1.000 | 0.562 | 0.099 | 0.970 | 0.977 | 0.730 | 0.335 | 0.035 | 0.009 |
| | 1.000 | 1.000 | 0.143 | 0.977 | 0.983 | 0.803 | 0.599 | 0.290 | 0.096 |
| Three-Party Polling (Dual SCM) | 0.974 | 0.391 | 0.056 | 1.000 | 0.939 | 0.660 | 0.218 | 0.033 | 0.019 |
| | 0.986 | 0.543 | 0.096 | 1.000 | 0.955 | 0.753 | 0.273 | 0.103 | 0.059 |

***Slides Outlining Four Issues Shared With Sun
Based on Exploring Correctness of our Jini Model***

SUs Registered for Notifications Should Receive Them

Sample Consistency Condition #4 (Service Notification)

For All (SM, SD, SCM, SU, NR):

(SU, NR) IsElementOf SCM requested-notifications &

(SM, SD) IsElementOf SCM registered-services &

Matches ((SM, SD), (SU, NR))

implies (SM, SD) IsElementOf (SU matched-services)*

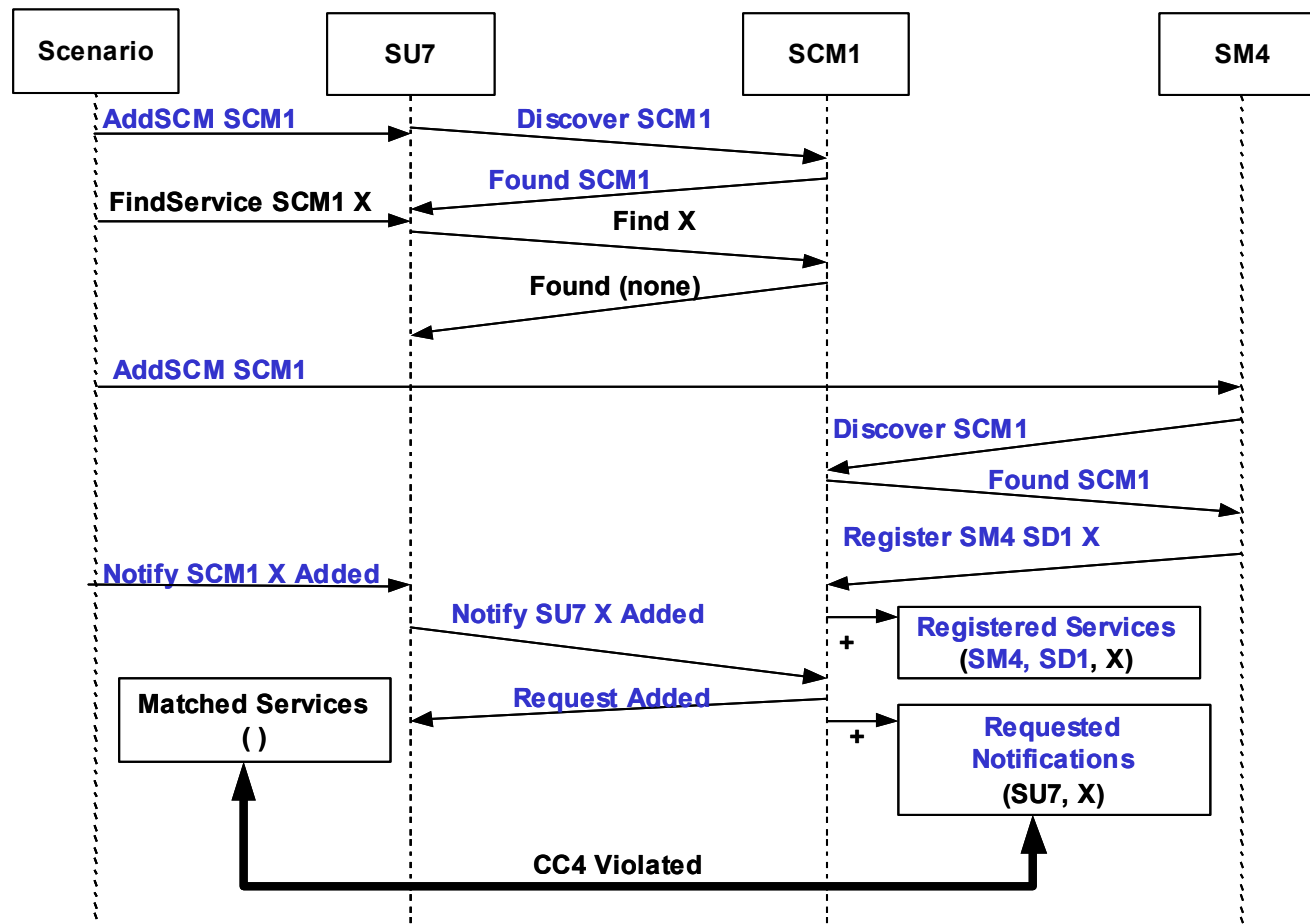
...that is, if an SU has requested notification with a Service Cache Manager of a service that matches a service description registered by a Service Manager on the same Cache Manager, then that service description should be provided to the Service User.

***Assuming absence of network failure and normal delays due to updates**

- SM is Service Manager
- SD is Service Description
- SCM is Service Cache Manager
- SU is Service User
- NR is Notification Request

- requested-notifications is a set of (SU,NR) pairs maintained by the SCM
- registered-services is a set of (SM,SD) pairs maintained by the SCM
- matched-services is the set of (SM,SD) pairs maintained by the SU

Should the Jini Specification Advise about Possibility for Registration Race Condition?



For All (SM, SD, SCM, SU, NR):
 (SU, NR) IsElementOf SCM requested-notifications & (CC4)
 (SM, SD) IsElementOf SCM registered-services &
 Matches((SM, SD), (SU, NR))
 implies (SM, SD) IsElementOf SU matched-services

Discovered SCMs Should Be Members of Same Group As Discovering Entity

Sample Consistency Condition #3 (Intersecting Group Membership)

For All (SM, SD, SCM):

SCM IsElementOf SM discovered-SCMs

(SM, SD) IsElementOf SCM registered-services

NOT (SCM IsElementOf SM persistent-list)

implies Intersection (SM GroupsToJoin, SCM GroupsMemberOf)*

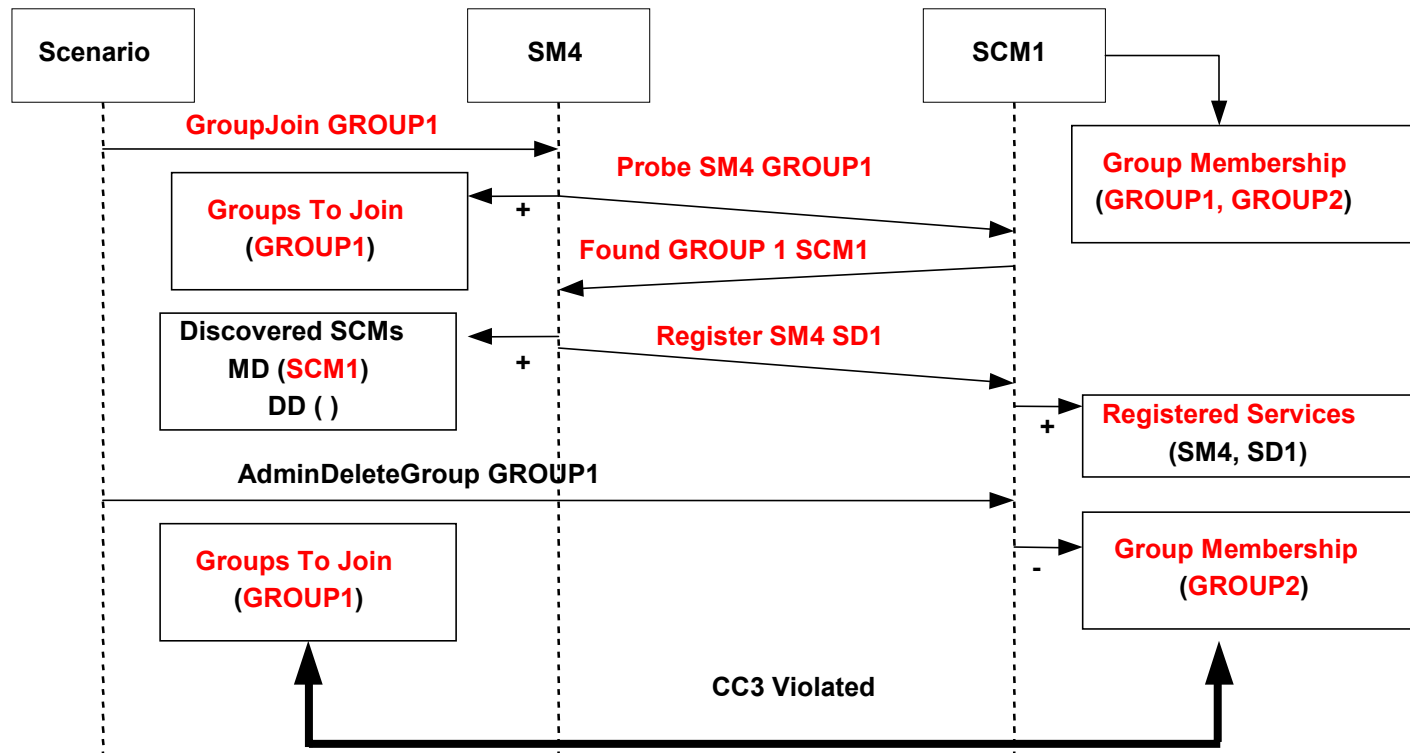
...that is, if a Service Manager has discovered, and registered its service descriptions on, a Service Cache Manager that is not on the Service Manager's persistent list, then the Service Manager must be seeking group membership in at least one group the Service Cache Manager belongs to.

***Assuming absence of network failure and normal delays due to updates**

- SM is Service Manager
- SD is Service Description
- SCM is Service Cache Manager

- registered-services is a set of (SM,SD) pairs maintained by the SCM
- discovered-SCMs is a set of SCMs discovered by the SM
- Persistent-list is the set of SCMs the SM is seeking through directed discovery

What Might Happen When SCM Changes Group Membership Dynamically?



For All (SM, SD, SCM):
 SCM IsElementOf SM discovered-SCMs & (CC3)
 (SM, SD) IsElementOf SCM registered-services &
 NOT (SCM IsElementOf SM persistent-list)
 implies Intersection (SM GroupsToJoin, SCM GroupsMemberOf)

Jini Entities Should Only Register with Discovered SCMs

Sample Consistency Condition #1(Register Only with Discovered SCMs)

***For All (SM, SD, SCM): (SM, SD) IsElementOf registered-services
implies SCM IsElementOf discovered-SCMs****

...that is, a Service Manager should register its Services on an Service Cache Manager only if it maintains that Cache Manager on its “known SCM” List.

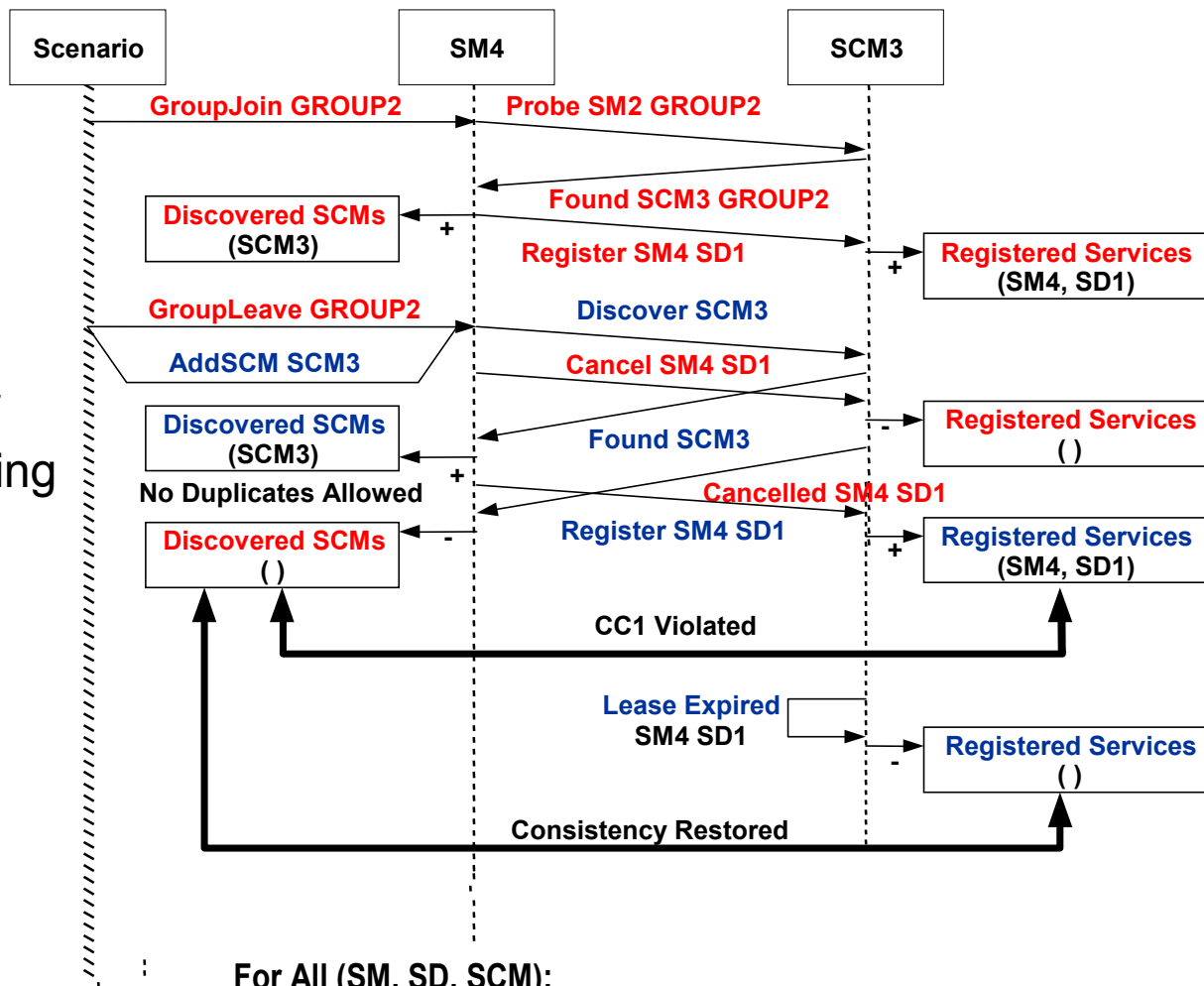
***Assuming absence of network failure and normal delays due to updates**

- SM is Service Manager
- SD is Service Description
- SCM is Service Cache Manager

- registered-services is a set of (SM,SD) pairs maintained by the SCM
- discovered-SCMs is a set of SCMs discovered by the SM

Could the Jini Specification Lead to Implementations Exhibiting Undesired Interaction between Directed and Multicast Discovery?

Based on one possible interpretation of specification using a *single-list assumption*



Jini Entities Should Register with All Discovered SCMs about which they maintain knowledge

Sample Consistency Condition #2 (Register Service with Discovered SCMs)

For All (SM, SD, SCM):

SCM IsElementOf SM discovered-SCMs &

SD IsElementOf SM managed-services

implies (SM, SD) IsElementOf SCM registered-services*

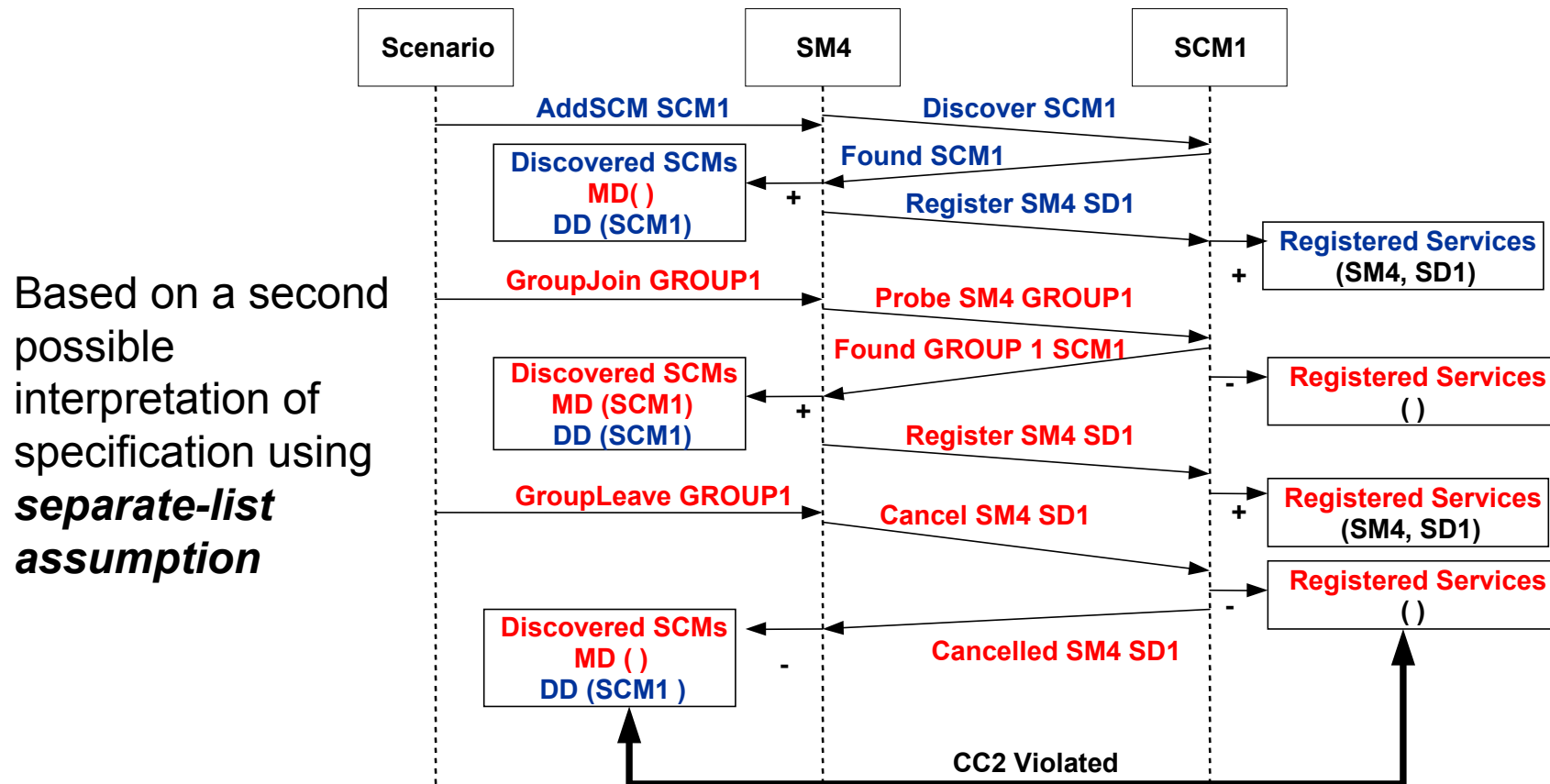
...that is, a Service Manager should register its Services on an Service Cache Manager if the Service Manager has discovered the Cache Manager and is maintaining the SCM identifier on its “known SCM” List.

***Assuming absence of network failure and normal delays due to updates**

- SM is Service Manager
- SD is Service Description
- SCM is Service Cache Manager

- discovered-SCMs is a set of SCMs discovered by the SM
- managed-services is a set of (SM,SD) pairs maintained by the SM
- registered-services is the set of (SM, SD) pairs maintained by the SCM

Could the Jini Specification Lead to Implementations Exhibiting Undesired Interaction between Directed and Multicast Discovery?



For All (SM, SD, SCM):
 SCM IsElementOf SM discovered-SCMs & (CC2)
 (SD) IsElementOf SM managed-services
 implies (SM, SD) IsElementOf SCM registered-services

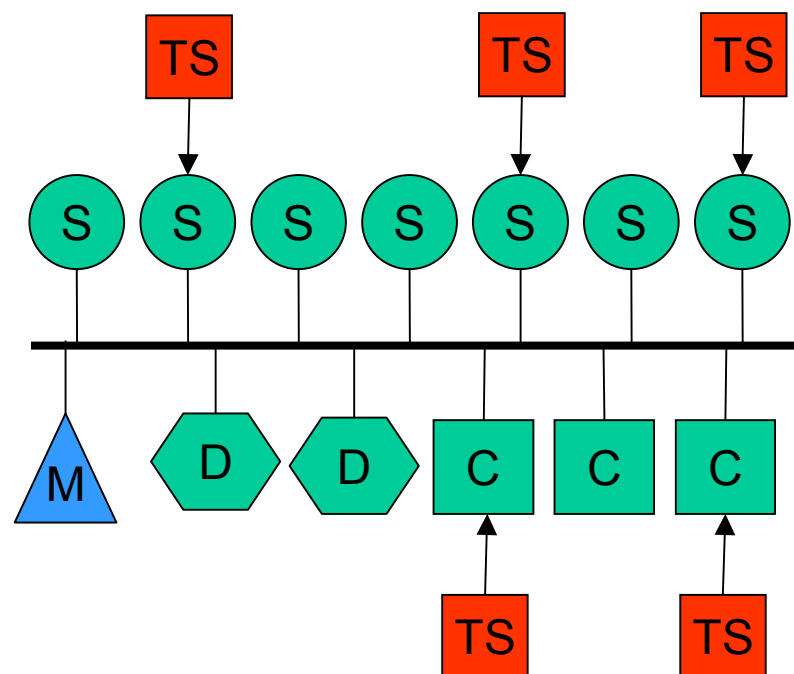
Slides Describing Performance Measurement Results from Jini and UPnP Implementations

Measurement Methodology

- **SDP Architectures & Protocols Differ Considerably**
 - Directory based vs flat peer-to-peer (combinations)
 - Java RMI and Serialized Objects vs. HTTP/SOAP/GENA and XML
- **How to reasonably compare the performance of such diverse technologies?**
- **Usage-Based Scenarios & Metrics**
 - Service initiation – restart, auto configuration, advertisement, renewal
 - Client active query – restart, by name, single instance, multiple instances, all instances
 - Client passive monitoring – persistent query for new instances, network restart
 - Event Notification and control – registration latency, notification latency, distributed control performance (control + event notification).
- **SDP-Independent Benchmark Service –**
 - Simple counting device/service that can be used to exercise all significant discovery/control capabilities of Jini and UPnP.
 - Supports – get/set service, GUIDs/attributes/type, control, event notification, GUI
- **Implementation-Independent Measurement Tools.**
 - Measure on-the-wire
 - Response/Load

Synthetic Workload Generation Tools

- **Objective** – Emulate large, dynamic environments of 100's of devices/services and 10's of control points / clients.
 - Dynamic devices providing the benchmark service.
 - Scripted control points execute measurement scenarios.
- **SDP Experimenters Toolkits**
 - Drive real SDP implementations
 - Emulate the behavior of a large number of dynamic devices
 - Emulate the behavior control points/ scripted behavior for testing
- **Jini & UPnP Initial development complete**
 - SunMS Jini, Intel UPnP on Linux platforms.
 - Target of 100's of devices and 10's of control points met.



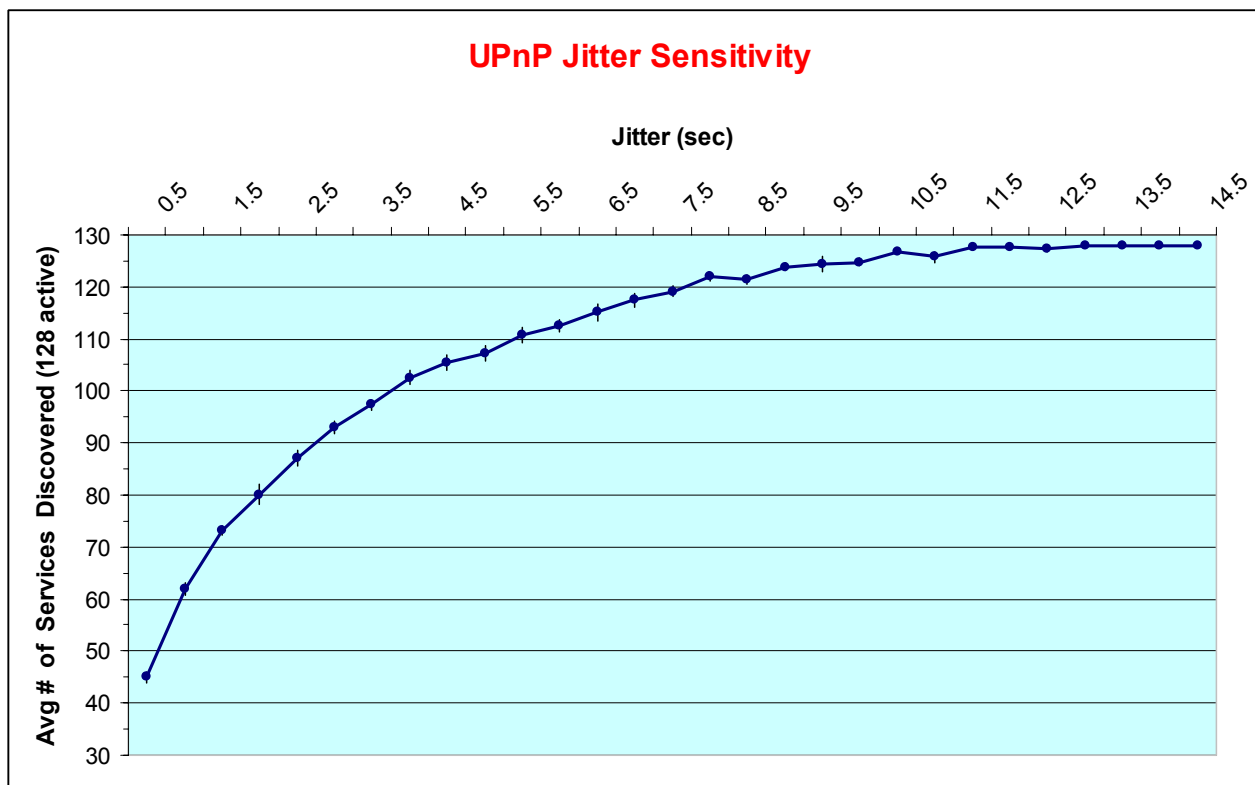
Sample Measurement Results

Notes:

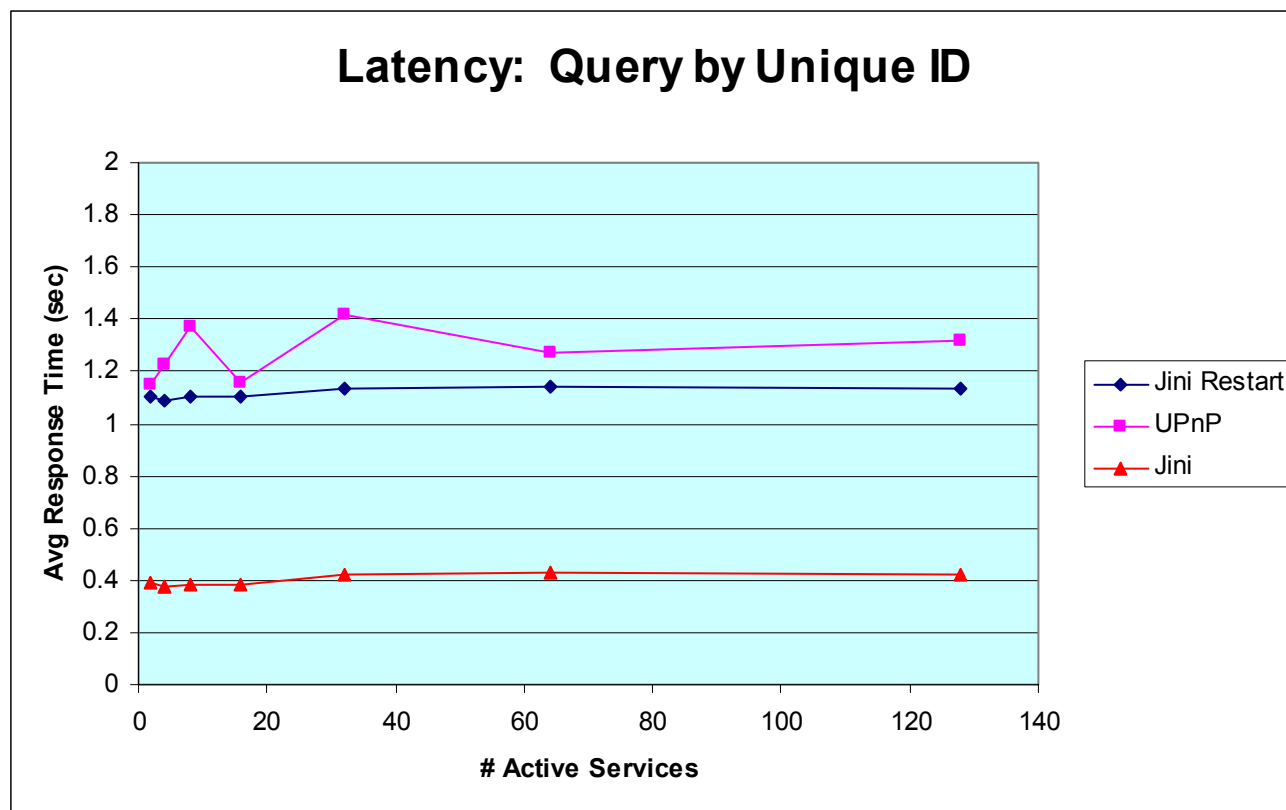
- Example results of Client Active Query scenarios.
- Query experiments measure latency / load through the client download of the service proxy / description.
- Results show two scenarios for Jini
 - Jini Restart – includes overhead of client discovery and first access of look up server.
 - Jini – query from “warm” client, after lookup server Discovery has completed.
- Default UPnP jitter value is 2.5 seconds
 - In device poll experiment, jitter values of 5.5 sec (64 services) and 11 sec (128 services) were used.

Linux UPnP Scaling Problems

- Problems encountered in achieving initial scaling goals for device emulation tools.
 - UPnP scalability above 40 devices a function of protocol tuning parameters (e.g., response jitter, multicast retransmission factor).
 - Errors in implementation of jitter algorithms

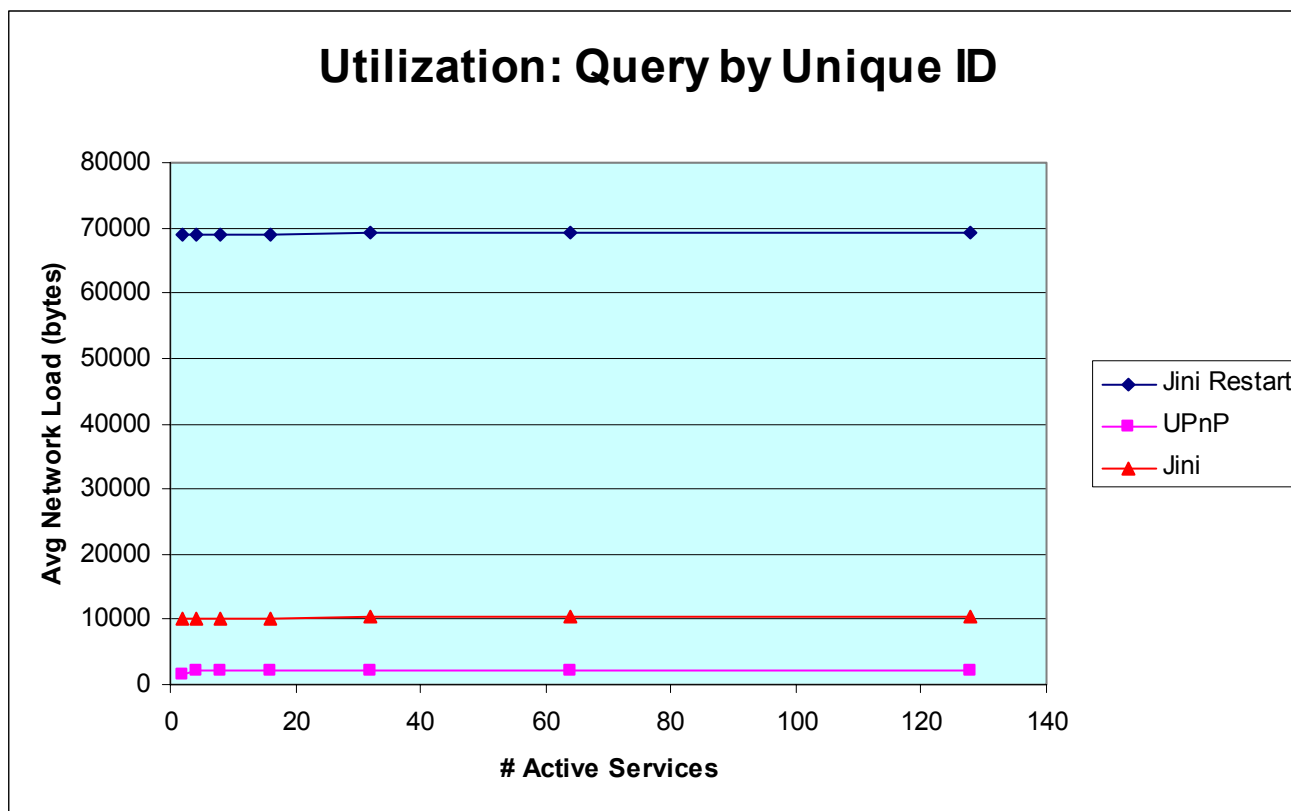


Some Example Results: Jini vs UPnP Discovery



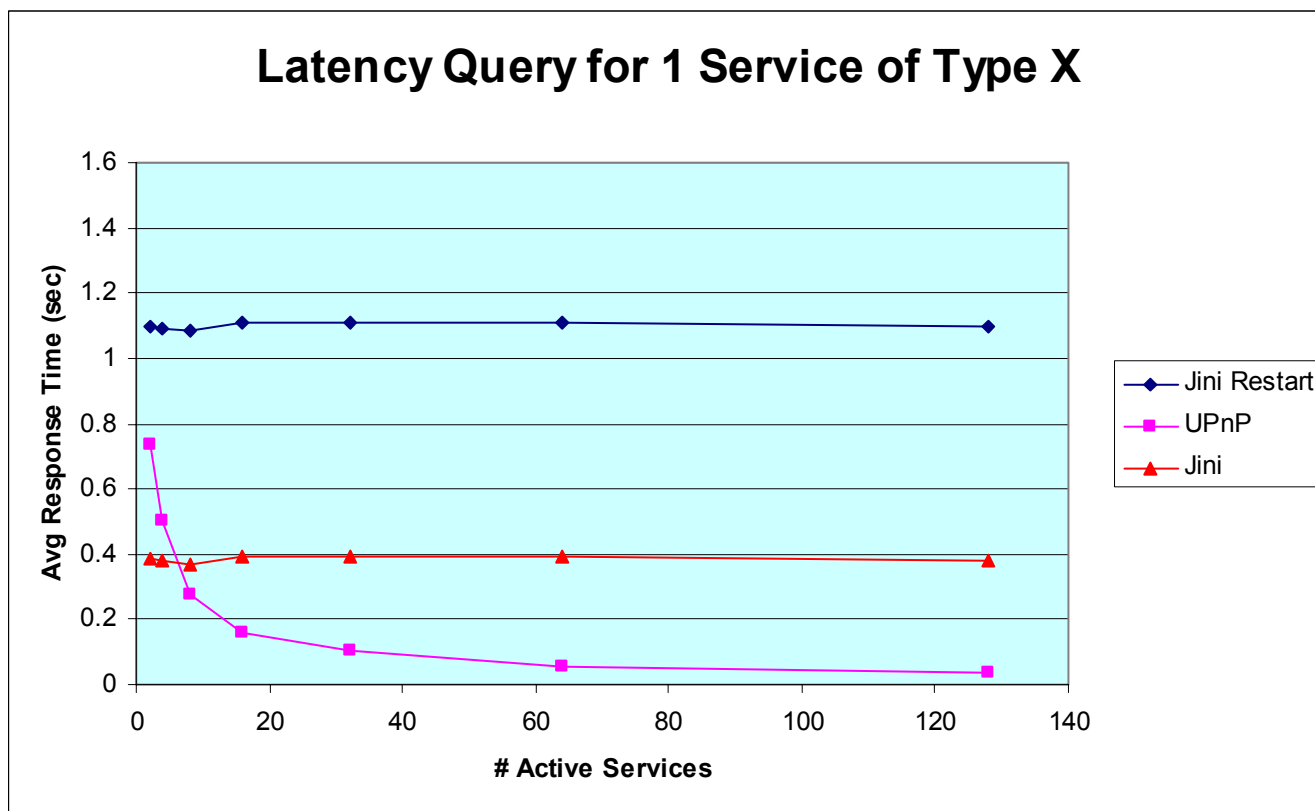
Performance of Query for Specific Service: “Find service X”

Some Example Results: Jini vs UPnP Discovery



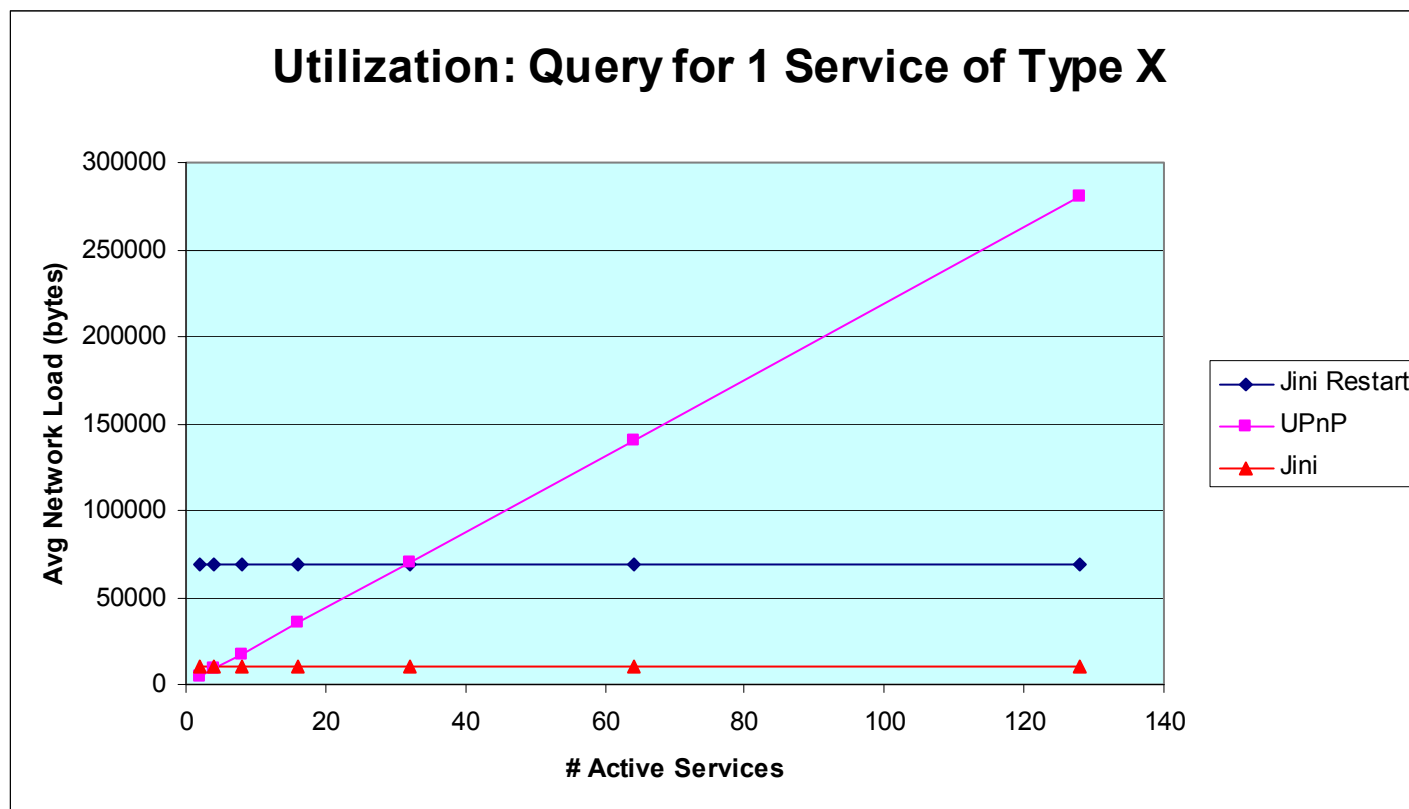
Performance of Query for Specific Service: “Find service X”

Some Example Results: Jini vs UPnP Discovery



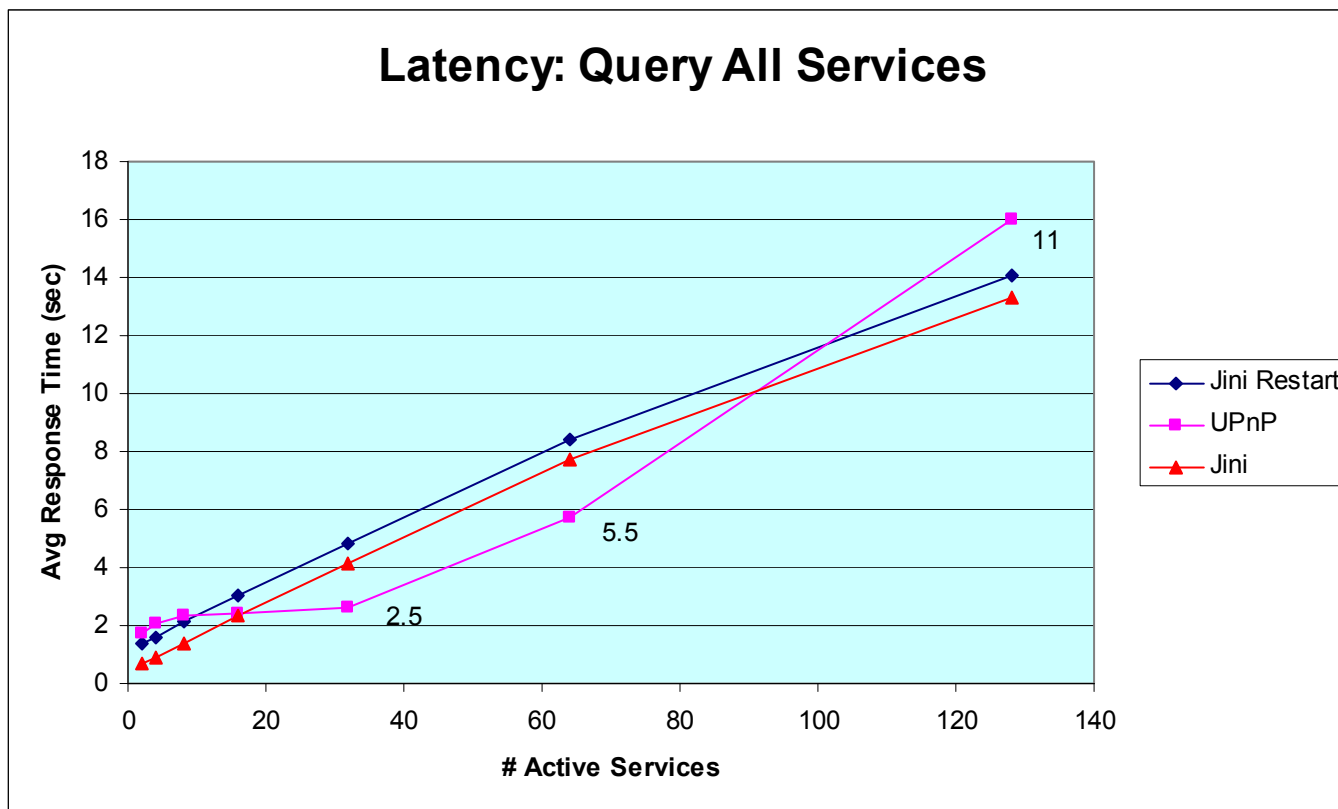
Performance of “anycast” Query: “Find one instance of type X”

Some Example Results: Jini vs UPnP Discovery



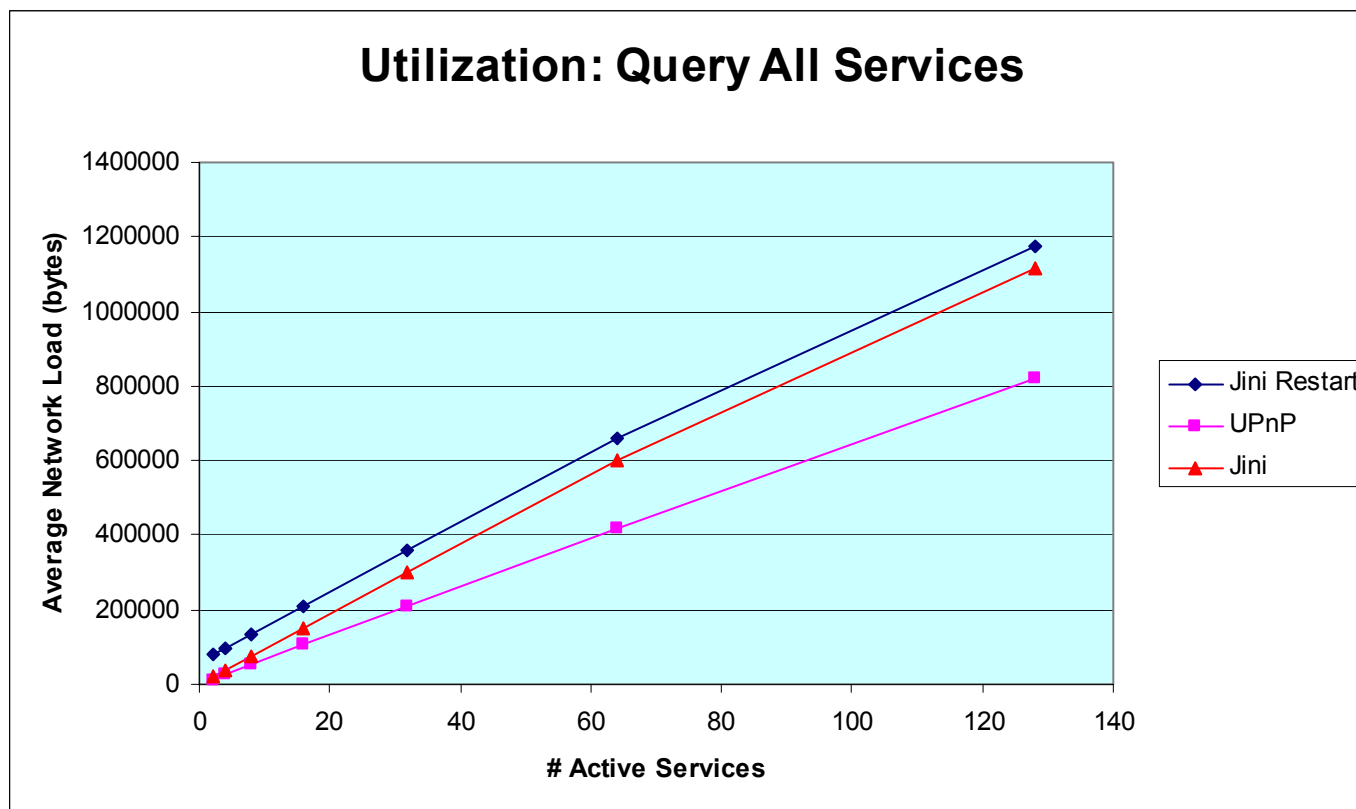
Performance of “anycast” Query: “Find one instance of type X”

Some Example Results: Jini vs UPnP Discovery



Performance of Service/Device Poll: “Find all active services”

Some Example Results: Jini vs UPnP Discovery



Performance of Service/Device Poll: “Find all active services”